

Estruturas de Dados - ESP412

Prof^a Ana Carolina Sokolonski

Bacharelado de Sistemas de Informação
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Campus de Feira de Santana

carolsoko@ifba.edu.br

November 26, 2024

Estruturas de Dados

1 Complexidade Computacional

2 Notação Big O

- Complexidade de Tempo
- Complexidade de Espaço

3 Referências

Complexidade Computacional

Complexidade Computacional

Definição

A complexidade computacional, ou complexidade algorítmica, é uma medida de desempenho que analisa o tempo de execução e uso de memória de um algoritmo em função do tamanho da entrada. Ela permite comparar a eficiência de algoritmos diferentes para resolver o mesmo problema.

Complexidade Computacional

Definição

A complexidade computacional, ou complexidade algorítmica, é uma medida de desempenho que analisa o tempo de execução e uso de memória de um algoritmo em função do tamanho da entrada. Ela permite comparar a eficiência de algoritmos diferentes para resolver o mesmo problema.

Entender a complexidade algorítmica é essencial para desenvolver programas eficientes, que consigam processar grandes volumes de dados sem consumir muitos recursos ou levar muito tempo.

Complexidade Computacional

Definição

A complexidade algorítmica avalia o desempenho de um algoritmo em relação ao tamanho da entrada, considerando o tempo de execução e uso de memória.

Complexidade Computacional

Definição

A complexidade algorítmica avalia o desempenho de um algoritmo em relação ao tamanho da entrada, considerando o tempo de execução e uso de memória.

Prever a velocidade de execução e a quantidade de memória consumida por um algoritmo para diferentes tamanhos de entrada é o objetivo da complexidade algorítmica.

Notação Big O

Notação Big O

Notação Big O

Uma das notações mais utilizadas para representar a **complexidade algorítmica** é a Notação Big O, que descreve o comportamento no pior caso (*worst case*) de um algoritmo.
[Cormen et al. 2009]

Notação Big O

Notação Big O

Uma das notações mais utilizadas para representar a **complexidade algorítmica** é a Notação Big O, que descreve o comportamento no pior caso (*worst case*) de um algoritmo. [Cormen et al. 2009]

A notação Big O é uma notação matemática que descreve o comportamento limitante de uma função quando o argumento tende a um valor específico ou ao infinito. Ela pertence a uma família de notações inventadas por Paul Bachmann, Edmund Landau e outros, coletivamente chamadas de notação Bachmann–Landau ou de notação assintótica. [D.E. Knuth 1973]

Notação Big O

Notação Big O

A notação Big O é uma forma de descrever a medida de quanta memória ou tempo computacional o algoritmo requer à medida que o tamanho da entrada aumenta. É uma ferramenta essencial para analisar e comparar algoritmos, permitindo-nos escolher a melhor solução para um determinado problema.

Notação Big O

$O(1)$: Tempo de execução constante

Independentemente do tamanho da entrada, o algoritmo sempre leva o mesmo tempo para ser executado. É o cenário ideal, onde a eficiência não é afetada pelo aumento dos dados. ex: acesso a um elemento num vetor

Notação Big O

$O(\log n)$: Tempo de execução logarítmico

O tempo de execução aumenta de forma logarítmica à medida que o tamanho da entrada aumenta. Algoritmos com essa complexidade são geralmente muito eficientes, como é o caso de árvores balanceadas e algoritmos de busca binária.

Notação Big O

$O(n)$: Tempo de execução linear

O tempo de execução cresce proporcionalmente ao tamanho da entrada. Algoritmos lineares percorrem cada elemento da entrada uma vez. Exemplos incluem busca linear (busca num vetor não ordenado) e algumas formas de ordenação.

Notação Big O

$O(n * \log n)$

O tempo de execução cresce proporcionalmente ao tamanho da entrada vezes o logaritmo. Exemplo: um algoritmo rápido de ordenação, como a ordenação *quicksort*.

Notação Big O

$O(n^2)$: Tempo de execução quadrático

O tempo de execução é o quadrado do tamanho da entrada. Algoritmos com essa complexidade são menos eficientes e podem levar muito tempo para processar grandes conjuntos de dados. Exemplos comuns são algoritmos de ordenação como o *Bubble Sort* e o *Insertion Sort*.

Notação Big O

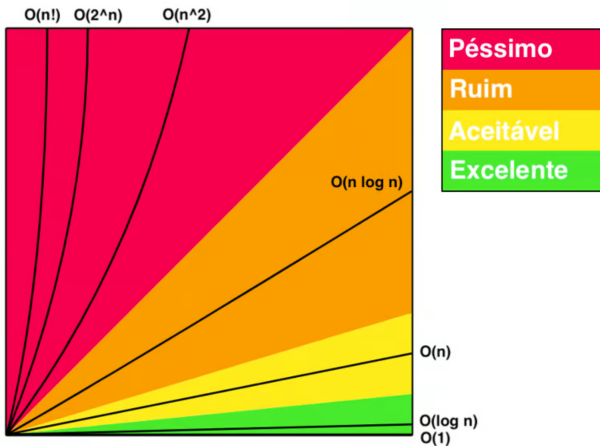
$O(n!)$: Tempo de execução fatorial

O tempo de execução cresce de acordo com o fatorial do número. Estes algoritmos são muito custosos e pouquíssimo eficientes. Exemplo: um algoritmo bastante lento, como o do caixeiro-viajante. O problema do caixeiro viajante é um desafio clássico de otimização combinatória, em que o objetivo é encontrar o menor caminho possível que passe por todas as cidades (ou pontos) visitando cada uma delas exatamente uma vez e retornando à cidade de origem (grafos).

Notação Big O

NOTAÇÃO BIG O	ALGORITMO DE EXEMPLO
$O(\log n)$	Busca binária
$O(n)$	Busca simples
$O(n * \log n)$	Ordenação Quicksort
$O(n^2)$	Ordenação de seleção
$O(n!)$	Problema do caixeiro viajante

Notação Big O



Complexidade de Tempo

Complexidade de Tempo

Complexidade de Tempo

Quando falamos em **complexidade de tempo algorítmica**, estamos nos referindo ao **tempo máximo que o algoritmo leva para executar dado um determinado tamanho de entrada**.

Complexidade de Tempo

Complexidade de Tempo

Quando falamos em **complexidade de tempo algorítmica**, estamos nos referindo ao **tempo máximo que o algoritmo leva para executar dado um determinado tamanho de entrada**.

A notação Big O é uma notação matemática que descreve o comportamento limitante de uma função quando o argumento tende a um valor específico ou ao infinito. Ela pertence a uma família de notações inventadas por Paul Bachmann, Edmund Landau e outros, coletivamente chamadas de notação Bachmann–Landau ou de notação assintótica.

Complexidade de Tempo

Complexidade de Tempo

Algoritmos clássicos na Ciência da Computação possuem diferentes complexidades no Big O, o que influencia diretamente em sua eficiência. Alguns algoritmos, como a pesquisa binária, possuem complexidades logarítmicas, enquanto outros, como o *selection sort*, têm complexidades quadráticas.

Complexidade de Tempo

Complexidade de Tempo

Algoritmos clássicos na Ciência da Computação possuem diferentes complexidades no Big O, o que influencia diretamente em sua eficiência. Alguns algoritmos, como a pesquisa binária, possuem complexidades logarítmicas, enquanto outros, como o *selection sort*, têm complexidades quadráticas.

Existem diversas práticas que podem ser adotadas para melhorar a complexidade algorítmica de um algoritmo ou programa. Reduzir trabalho desnecessário, evitar soluções exponenciais e escolher bons algoritmos são algumas das dicas que podem proporcionar melhor performance e escalabilidade.

Complexidade de Tempo

Como calcular a Notação Big O?

Algoritmos clássicos na Ciência da Computação possuem diferentes complexidades no Big O, o que influencia diretamente em sua eficiência. Alguns algoritmos, como a pesquisa binária, possuem complexidades logarítmicas, enquanto outros, como o *selection sort*, têm complexidades quadráticas.

Complexidade de Tempo

Como calcular a Notação Big O?

Algoritmos clássicos na Ciência da Computação possuem diferentes complexidades no Big O, o que influencia diretamente em sua eficiência. Alguns algoritmos, como a pesquisa binária, possuem complexidades logarítmicas, enquanto outros, como o *selection sort*, têm complexidades quadráticas.

Para calcular o Big O temos que analisar quantas vezes o algoritmo executa, de acordo com o tamanho da entrada, no pior caso. Por exemplo, suponha uma busca em um vetor desordenado, o pior caso será quando o elemento buscado estiver na última posição visitada, neste caso, você terá que testar todos os elementos para achá-lo, dessa forma, o custo será n , portanto o algoritmo tem custo no pior caso de $O(n)$.

Complexidade de Tempo

Como calcular a Notação Big O?

Suponhamos agora que o vetor está ordenado, assim, podemos aplicar o algoritmo de Busca Binária, onde iniciamos a busca no meio do vetor, avaliamos se o elemento é maior, ou menor que o meio, e buscamos apenas a metade que importa, descartando a cada passo metade da entrada... assim o algoritmo terá, no pior caso, que testar $\log_2 n$, portanto o algoritmo tem custo no pior caso de $O(\log_2 n)$, ou simplesmente, $O(\log n)$.

Complexidade de Espaço

Complexidade de Espaço

Complexidade de Espaço

Devemos destacar que a Notação Big O vista até aqui refere-se à complexidade de tempo computacional. Nesta notação, a complexidade do tempo do Big O descreve o tempo de execução no pior caso.

Complexidade de Espaço

Complexidade de Espaço

Devemos destacar que a Notação Big O vista até aqui refere-se à complexidade de tempo computacional. Nesta notação, a complexidade do tempo do Big O descreve o tempo de execução no pior caso.

A complexidade de espaço Big O descreve quanta memória é necessária para executar um algoritmo no pior cenário, assim ela é relacionada com o quanto de memória o programa usará e, portanto, também é um fator importante a ser analisado.



Complexidade de Espaço

Complexidade de Espaço

A complexidade de espaço funciona de modo semelhante à complexidade de tempo. Por exemplo, a *selection sort* tem uma complexidade de espaço de $O(1)$, porque ela somente armazena um valor mínimo e seu índice para comparação, o espaço máximo usado não aumenta com o tamanho da entrada.

Referências

Referências

-  CORMEN, T. H. et al. *Introduction to Algorithms*. 2nd. ed. [S.l.]: The MIT Press, 2009. ISBN 0262032937.
-  D.E. Knuth. *The Art of Computer Programming*. [S.l.]: Addison-Wesley, 1973. v. 1 - 3.